Attorney Docket No.: 20699-000510US
Client Reference No.: 50P3902.01

PATENT APPLICATION

APPLICATION PROGRAMMING INTERFACE FOR COMMUNICATION BETWEEN AUDIO/VIDEO FILE SYSTEM AND AUDIO/VIDEO CONTROLLER

Inventor(s):

Ibrahim Cem Duruoz, a citizen of Turkey, residing at,
1241 20th Avenue, Apt 6
San Francisco, CA 94122

Kosuke Nakai, a citizen of Japan, residing at,
7-23-201 Higashi-Terao, Nakadai,
Tsurumi-Ku, Yokohama
Kanagawa, 230-0017
Japan

Natarajan Sundaresan, a citizen of India, residing at,
835 Bing Drive #5
Santa Clara, CA 95051

Assignee(s):

Sony Corporation
7-35 Kitashinagawa,
6-Chome
Shinagawa-Ku
Tokyo, Japan

Sony Electronics Inc.
One Sony Drive,
Park Ridge, New Jersey 07656

Entity: Large

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111-3834
Tel: 415-576-0200

APPLICATION PROGRAMMING INTERFACE FOR COMMUNICATION BETWEEN AUDIO/VIDEO FILE SYSTEM AND AUDIO/VIDEO CONTROLLER

CROSS-REFERENCES TO RELATED APPLICATION(S)

[01] The present application claims the benefit of priority under 35 U.S.C. § 119 from U.S. Provisional Patent Application Serial No. 60/272,863, entitled "APPLICATION PROGRAMMING INTERFACE FOR COMMUNICATION BETWEEN AUDIO/VIDEO FILE SYSTEM AND AUDIO/VIDEO CONTROLLER" by Duruoz et al., filed March 1, 2001, the disclosure of which is hereby incorporated by reference in its entirety for all purposes.

[02] The present application is related to the following co-pending, commonly assigned applications including: U.S. Provisional Patent Application Serial No. 60/272,798, entitled "APPLICATION PROGRAMMING INTERFACE FOR COMMUNICATION BETWEEN AUDIO/VIDEO FILE SYSTEM AND HARD DISK DRIVE SOFTWARE" by Duruoz et al., filed on March 1, 2001; U.S. Patent Application Serial No. (to be assigned) entitled "PROCESS CONTROL MANAGER FOR AUDIO/VIDEO FILE SYSTEM" by Duruoz, filed concurrently herewith, which claims the benefit of priority from U.S. Provisional Patent Application Serial No. 60/272,804, entitled "PROCESS CONTROL MANAGER FOR AUDIO/VIDEO FILE SYSTEM" by Duruoz, filed on March 1, 2001; and U.S. Patent Application Serial No. (to be assigned) entitled "METHOD AND SYSTEM FOR OPTIMIZING DATA STORAGE AND RETRIEVAL BY AN AUDIO/VIDEO FILE SYSTEM USING HIERARCHICAL FILE ALLOCATION AND OTHER TABLES" by Duruoz, filed concurrently herewith, which claims the benefit of priority from U.S. Provisional Patent Application Serial No. 60/272,864, entitled "METHOD AND SYSTEM FOR OPTIMIZING DATA STORAGE AND RETRIEVAL BY AN AUDIO/VIDEO FILE SYSTEM USING HIERARCHICAL FILE ALLOCATION AND OTHER TABLES" by Duruoz, filed on March 1, 2001, all of which are incorporated in their entirety by reference herein.

BACKGROUND OF THE INVENTION

[03] With the proliferation of digital audio and video devices, it has become necessary to establish a high speed serial communication mechanism that is capable of

allowing efficient and fast transfer of audio/video (A/V) data between devices. IEEE 1394 has so far been a successful candidate for this purpose.

[04] While the transmission of A/V data using industry standard such as IEEE 1394 is well known, the storage of A/V data on modern computing devices is becoming an important issue. Modern computing devices are rapidly moving toward offering various kinds of functionality and features in an integrated manner. For example, a personal computer (PC) nowadays generally contains a myriad of computer programs which offer a variety of functionality, such as word processing programs as well as computer programs which are capable of processing and playing video and audio data. As a result, there is a need to store both A/V data and non-A/V data and both types of data are typically stored on a computer hard disk together.

[05] To store data on the hard disk or other media, a file system is generally required to be in place to organize files and accomplish a number of file-handling tasks such as file creation, deletion, access, save and update. Furthermore, it is preferable and common practice for a file system to have a directory structure which organizes and provides information about all the files stored in the file system. Typically, a file system is capable of storing both A/V and non-A/V data.

[06] A/V data and non-A/V data, however, are fundamentally different in a number of respects. For example, the size of A/V data files is relatively large when compared to the size of non-A/V data files. As a result, handling files of varying sizes in an efficient manner becomes a challenging issue.

[07] Moreover, since A/V data files are generally very large, it is usually not possible to store an entire A/V data file in a temporary storage area such as the random access memory (RAM). Therefore, due to the large size of an A/V data file, efficient random access of A/V data within the A/V data file also presents a problem.

[08] Furthermore, the retrieval and processing of A/V data are subject to relatively stringent time and order-related constraints. Such constraints are necessary to prevent the corruption of A/V data and avoid potential glitches. For example, in order to provide a smooth presentation of video images, A/V data need to be retrieved and processed quickly and continuously in the correct sequential order. Otherwise, the video images might become choppy and incomplete. Hence, it would be desirable to develop and provide an application programming interface between an audio/video controller and a file system capable of handling a mix of A/V and non-A/V data so as to allow A/V data to be stored, retrieved and processed in an efficient manner.

[09] Moreover, changes made to the internal mechanics of a file system should ideally be apparent to any external devices that need to communicate with the file system. This would obviate the need to make changes to the external devices every time the internal mechanics of the file system is changed. Thus, it would also be desirable to develop and provide an application programming interface between an audio/video controller and a file system so that the audio/video controller is not affected by changes to the file system.

SUMMARY OF THE INVENTION

[10] The present invention is directed to an application programming interface (API). More specifically, the present invention relates to an application programming interface designed for handling interaction between an audio/video controller and an audio/video file system.

[11] In an exemplary embodiment, the application programming interface of the present invention includes a number of function calls which direct the audio/video file system to perform a variety of functions. These function calls are described as follows.

[12] A first group of function calls is designed to handle smaller files such as a non-A/V descriptor file. This group of function calls includes a load function call, a store function call, a delete function call, and a validity function call. The load function call is provided to direct the audio/video file system to retrieve desired descriptor information from a storage medium such as a computer hard disk. The store function call is provided to direct the audio/video file system to store specified descriptor information onto the storage medium. The delete function call is provided to direct the audio/video file system to delete specified descriptor information from the storage medium. The validity function call is provided to verify the validity of a specified descriptor.

[13] A second group of function calls is designed to handle larger files such as an A/V file. This group of function calls includes a play function call, a record function call, a stop function call, a pause function call, a resume function call, and an address retrieval function call. The play function call is provided to cause a specified file to be played. The record function call is provided to cause specified data to be recorded onto the storage medium. The stop function call is provided to cause a play or record operation to be stopped. The pause function call is provided to cause a play or record operation to be paused. The resume function call is provided to cause a previously paused operation to resume. The address retrieval function call is provided to determine a logical block address of a specified file.

[14] This second group of function calls further includes additional function calls which allow certain trick plays to be performed, including a fast forward function call, a slow forward function call, a step forward function call, a fast reverse function call, a slow reverse function call, and a step reverse function call. The fast forward function call is provided to cause a data stream to be fast-forwarded. The slow forward function call is provided to cause a data stream to move forward frame-by-frame, or by certain specified increment, with each displayed frame being displayed repeatedly based on a predetermined duration. The step forward function call is provided to cause a data stream to move forward by one or more frames and then return to a pause state. Similarly, the fast reverse function call is provided to cause a data stream to be fast-reversed. The slow reverse function call is provided to cause a data stream to move backward frame-by-frame, or by certain specified increment, with each displayed frame being displayed repeatedly based on a predetermined duration. The step reverse function call is provided to cause a data stream to move backward by one or more frames and then return to a pause state.

[15] The API of the present invention provides a number of advantages. For example, the API supports the handling of large and small files thereby allowing A/V data files and other types of data files to be processed efficiently. The API also supports a variety of features within a file system including scalability to large disks, directory structures and delete, add, record, play and other trick play operations.

[16] Furthermore, the API allows several A/V data streams or the same A/V data stream to be recorded or played concurrently. The API further supports record and play operations starting from within an A/V data file thereby providing more efficient random data access.

[17] Moreover, the API optimizes disk access thereby speeding up the retrieval and handling of data.

[18] In addition, the API permits devices each capable of handling a different type of data, such as isochronous and asynchronous data, to communicate with the audio/video file system. For example, the API can be used by an audio/video controller capable of processing packets transported using the 61883 protocol as well as a serial-bus-protocol-2 controller.

[19] Reference to the remaining portions of the specification, including the drawings and claims, will realize other features and advantages of the present invention. Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with

respect to accompanying drawings. In the drawings, like reference numbers indicate identical or functionally similar elements.

BRIEF DESCRIPTION OF THE DRAWINGS

[20] Fig. 1 is a simplified schematic functional block diagram showing the application programming interface in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[21] Exemplary embodiments of the present invention will now be described. Fig. 1 is a simplified schematic functional block diagram showing the application programming interface in accordance with the present invention. Fig. 1 also shows the system architecture of an audio/video (A/V) system 10. The A/V system 10 includes a 1394 cable 12, a protocol interpreter 14, an A/V command queue 16, an A/V controller 18, an A/V file system 20, a SBP-2 controller 22, a cache control 24 and a high-level I/O control 26 and a physical I/O control 28.

[22] The 1394 cable 12 is used to transmit A/V data to and from another device (not shown), such as a video recorder, to the A/V system 10. The 1394 cable 12 is connected to the protocol interpreter 14. A function of the protocol interpreter 14 is to decode an incoming packet received from the 1394 cable 12 and then determine the identity of the transport protocol for such packet. More specifically, the protocol interpreter 14 determines whether the transport protocol for a transmitted packet is the 61883 protocol or the serial-bus-protocol-2 (SBP-2). These two protocols are used to define the data contained in a transmitted packet. The 61883 protocol provides for isochronous transmission of data packets which are usually used for time-critical data, such as video and audio data, as well as asynchronous transmission of control packets; whereas, protocol SBP-2 provides for asynchronous transmission which is generally used for data that are more tolerant of delay, such as text and electronic mails. Typically, the protocol interpreter 14 is implemented using both hardware and software.

[23] Upon determining the identity of the transport protocol for each packet, the protocol interpreter 14 accordingly routes the commands carried by that packet to either the A/V command queue 16 or the SBP-2 controller 22 for further processing. Commands carried by packets which use the 61883 protocol as the transport protocol are routed to the A/V command queue 16, while commands carried by packets which use the

SBP-2 protocol are delivered to the SBP-2 controller 22. The A/V data contained in the packets are separately routed to the storage media, such as a hard disk, directly.

[24] A function of the A/V command queue 16 is to provide a staging area to queue up any commands that are awaiting execution. Commands which are ready to be executed are then passed to the A/V controller 18 for execution.

[25] A function of the A/V controller 18 is to determine whether a command to be executed need to invoke the A/V file system 20. In other words, whether the A/V file system 20 is required to perform certain functions pursuant to the command. If it is determined that the A/V file system 20 is to be invoked, the A/V controller 18 uses the appropriate API to interact with the A/V file system 20. Details of the API will be further described below.

[26] The function of the A/V file system 20 is to organize A/V data in such a manner so as to allow efficient data processing and retrieval. In general, the A/V file system 20 organizes the data into files or objects. A descriptor is used to store and provide information on the properties of a specific object or a list of objects. Examples of various types of descriptors used in connection with the API of the present invention are listed in Table 1 below.

[27] In an exemplary embodiment, the API of the present invention interfaces with an audio/video file system. For example, the API of the present invention can be used to interface with A/V file systems such as BSD FFS, Linux ext2, Macintosh HFS, and Windows NT's NTFS etc.

[28] The various types of descriptors listed in Table 1 will now be described. A descriptor of the type "object entry" contains information about an object such as objectID. Each object is assigned a unique objectID so as to allow an object to be identified on an individual basis. For example, if an object contains data for a movie, then the descriptor for that object may contain information about that movie, such as movie length, title, size (e.g., in bytes), and other relevant information etc.

[29] A descriptor of the type "contents list" contains information about a list of objects or a list of one or more lists of objects. Each "contents list" is assigned a unique ListID so as to allow a "contents list" to be uniquely identified. For example, assuming there are a number of objects each containing data for a particular movie, a "contents list" descriptor may contain information about a list of objects (i.e. movies) that fall into a particular category such as action or drama. In another example, a descriptor may contain information about a list of one or more lists of movies. The initial or root list may

contain lists of movies from a particular time period like the 1990's and each list contains movies of a particular category from that time period.

[30] A descriptor of the type "performance list" contains information about a list of objects which are related to each another for performance or execution purposes. Similarly, each "performance list" descriptor is assigned a unique ListID so as to allow a particular "performance list" to be uniquely identified. For example, if a movie contains a number of previews, this descriptor may contain information about a list of such previews for that particular movie.

[31] Descriptors of the types "SID" and "DSSD" are used to provide information about the physical storage media accessible to the A/V file system. Via these two descriptors, external devices, such as a video recorder, can obtain information about the physical storage media. "SID" is an acronym for subunit identifier descriptor and "DSSD" is an acronym for disc subunit status descriptor.

[32] A descriptor of the type "HMS table" contains timing and content location information about an object. "HMS" stands for hour, minute and second. For example, if the object is a movie, then such descriptor contains information such as total length of the movie, start and end times of different segments within the movie, and specific disk locations of corresponding time points in the movie, etc.

[33] The use of these descriptors in the API will be explained more fully below.

TABLE 1

Descriptor Types	
Type	ID
Object entry	Object ID
Contents list	List ID
Performance list	List ID
SID	N/A
DSSD	N/A
HMS table	Object ID

[34] The A/V file system 20 also interacts with the high-level I/O control 26 and the cache control 24 to allow the physical storage media (not shown) to be manipulated.

[35] The A/V system 10, as shown in Fig. 1, generally operates in the following manner. A/V data, usually in the form of packets, are transmitted through the 1394 cable 12 in accordance with the IEEE 1394 protocol to the protocol interpreter 14. As it is commonly known and understood in the industry, the IEEE 1394 protocol is designed to allow high speed serial transmission of data.

[36] The protocol interpreter 14 then decodes the packets and identifies the pertinent transport protocol for each packet. If it is determined that the transport protocol for a packet is SBP-2, then the commands carried by the packet are processed by the SBP-2 controller 22 and the data are directly stored to or retrieved from the SBP-2 partition of the storage media (not shown) under the control of the physical I/O control 28.

[37] If the protocol interpreter 14 determines that the transport protocol for the packet is 61883 (which means the data component of the packet is transmitted on an isochronous basis and the control component of the packet is transmitted on an asynchronous basis), the commands embedded in the packet is inserted into the A/V command queue 16 to await execution. The A/V controller 18 then processes the commands from the A/V command queue 16 and responds to them depending on their content. Certain commands require functions to be performed by the A/V file system 20. Such functions are effectuated by using the appropriate API described below.

[38] Fig. 1 also shows an exemplary embodiment of the API of the present invention. As shown in Fig. 1, the logic for interpreting and processing the API is resident on the A/V file system 20. The API provides an interface for a driver or application, such as the A/V controller 18, to allow the A/V controller 18 to direct actions to be performed by the A/V file system 20.

[39] The API of the present invention provides the software interface between the A/V controller 18 and the A/V file system 20. This is done by providing an application programming library of function calls independent of the operating system or internal implementation which provides command, data transfer and other capabilities to the A/V controller 18. This layering enables the API to present a consistent interface to the A/V controller 18 and/or other devices even though the underlying structure of the A/V file system 20 should change.

[40] In an exemplary embodiment, the API of the present invention includes a number of operating-system-independent function calls to be made by an

operating-system-specific driver, such as the A/V controller 18. The A/V controller 18 uses the API to manipulate or otherwise interact with the A/V file system 20. Table 2 contains a list of these API function calls.

TABLE 2

API Function Call Summary	
Function	Purpose
LoadDescriptor ()	Retrieve descriptor information from media
StoreDescriptor ()	Store descriptor information onto media
DeleteDescriptor ()	Delete descriptor information from media
CheckValidityOfID ()	Check validity of specified ID value
Play ()	Play track or object
Record ()	Record track or object
Stop ()	Stop track or object under recording or playing
Pause ()	Pause current play or record operation
Resume ()	Start play or record previously paused operation
FastForward ()	Fast forward a track or object
SlowForward ()	Forward a track or object by certain specified increment
StepForward ()	Forward a track by one or more frames and return to pause state
FastReverse ()	Fast reverse a track or object
SlowReverse ()	Reverse a track or object by certain specified increment
StepReverse ()	Reverse a track by one or more frames and return to pause state
GetCurrentLBA ()	Retrieve logical byte address offset for recorded track or object

[41] Preferably, each function call of the API of the present invention is capable of passing parameters. Most function calls also return a result code to indicate the result of the function calls. Each function call will now be described in detail.

[42] A first group of function calls is designed to handle smaller files such as a descriptor file. This group of function calls includes the LoadDescriptor function call,

the StoreDescriptor function call, the DeleteDescriptor function call, and the CheckValidityofID function call.

[43] The LoadDescriptor function call is used to retrieve descriptor information from the storage media. A sample format for this function call is as follows:

LoadDescriptor (descID, type, offset, size, *p_dest, *cb)

---

[44] The parameters "descID" and "type" provide information relate to a specified descriptor. The parameter "descID" specifies the assigned identification number of the descriptor that is to be retrieved; and the parameter "type" specifies the type of descriptor that is to be retrieved. As shown in Table 1 above, there are various types of descriptors.

[45] The parameter "offset" specifies the start byte offset value to be used for loading the descriptor data. Preferably, this offset value is a multiple of sectors. A sector is broadly defined as the smallest addressable portion of a storage medium. For example, if the storage medium is a hard disk, then the sector size is typically 512 bytes. By specifying an "offset" value, a user is able to specify the position within the descriptor data where initial loading should begin. For example, if the descriptor is of the type "object entry" and the object entry represents a movie, a user then can, with the aid of this function call, specify the location where the desired descriptor information about the movie is located and then retrieve such information.

[46] The parameter "size" specifies the number of sectors that are to be loaded from the start byte offset which is provided by the parameter "offset." In other words, the quantity of data to be read beginning from the start byte offset can be specified.

[47] The parameter "*p_dest" represents the destination pointer for the descriptor data. Before the descriptor data is read from the storage media, a designated location in a temporary memory area, such as a RAM, is assigned by the A/V controller 18 to store such data for subsequent retrieval. The beginning position of this designated location is represented by the destination pointer so as to allow the A/V controller 18 to retrieve the data.

[48] The parameter "*cb" represents a pointer to a call back function to be used by the A/V file system 20 to signal to the program issuing the function call that processing is complete. For example, after the A/V controller 18 issues this function call to the A/V file system 20, the A/V controller 18 does not stay in contact with the A/V file system 20 to monitor its progress. When the A/V file system 20 has performed the desired function pursuant to the function call, the A/V file system 20 activates the call back function

provided by the A/V controller 18 thereby signaling to the A/V controller 18 that the desired function has been performed.

[49] For example, using the LoadDescriptor function call, the A/V controller 18 can direct the A/V file system 20 to retrieve selective data from an A/V data file stored on the storage media. Such A/V data file can represent, for example, a movie or other video presentation.

[50] It is to be noted that some of the function calls discussed below use the same parameters as explained in connection with the LoadDescriptor function call. These same parameters represent the same information or perform the same function for these other function calls.

[51] The StoreDescriptor function call is used to store descriptor information onto the storage media. A sample format for this function call is as follows:

StoreDescriptor (descID, type, offset, size, *p_data, *cb)

---

[52] As mentioned above, the parameters "descID" and "type" provide information relate to a specified descriptor.

[53] The parameter "offset" specifies the start byte offset value to be used for storing the descriptor data. Preferably, this offset value is a multiple of sectors. By specifying an "offset" value, a user is able to specify the position within the descriptor data where the storing should begin. For example, if the descriptor is of the type "object entry" and the object entry represents a movie, a user then can, with the aid of this function call, specify where to start storing additional descriptor information within the descriptor for that movie.

[54] The parameter "size" specifies the number of sectors that are to be stored from the start byte offset which is provided by the parameter "offset." In other words, the quantity of data to be stored beginning from the start byte offset can be specified.

[55] The parameter "*p_data" represents the destination pointer for the descriptor data. Before the descriptor data is stored onto the storage media, a designated location in a temporary memory area, such as a RAM, which is to contain the descriptor data to be stored is identified. The beginning position of this designated location is represented by the destination pointer so as to allow the A/V file system 20 to retrieve the data and then store such data onto the storage media.

[56] It should be further noted that this function call also serves to create a new descriptor to store the specified descriptor information if the specified descriptor does not exist.

[57] The DeleteDescriptor function call is used to delete descriptor information from the storage media. A sample format for this function call is as follows:

DeleteDescriptor (descID, type, *cb)

---

[58] Preferably, depending on the "type" of descriptor that is specified, additional operations are performed which are consistent with this function call. For example, if the "type" of descriptor specified is "object entry," then the corresponding object related to this descriptor is accordingly deleted from the physical storage media and other descriptors, such as content list and performance list, which contain information about the corresponding object are similarly updated accordingly.

[59] The CheckValidityOfID function call is used to check the validity of the specified ID value. A sample format for this function call is as follows:

CheckValidityOfID (descID, type, *cb)

---

[60] The parameters "descID" and "type" are supplied so that the A/V file system 20 can verify the validity of the descriptor. If the descriptor corresponding to the specified "descID" already exists on the storage media, the call back function returns a value of "TRUE," otherwise "FALSE" is returned.

[61] A second group of function calls is designed to handle larger files such as an A/V file. This group of function calls includes the Play function call, the Record function call, the Stop function call, the Pause function call, the Resume function call, and the GetCurrentLBA function call.

[62] The Play function call is used to play a specified track or object loaded from the storage media. A sample format for this function call is as follows:

Play (chanID, objectID, inPoint, outPoint, speed, type, *cb)

---

[63] The parameter "chanID" specifies which one of a number of channels is to be used to display the desired A/V data. In an exemplary embodiment where the 1394

cable 12 is used to transmit data, the 1394 cable 12 is able to accommodate one or more data streams simultaneously using multiple separate channels. In the case of protocol 1394, there is a maximum of sixty-three (63) channels. As a result, each recipient device is capable of displaying one of the data streams via a particular channel depending on how such recipient device is programmed. For example, if three movies are simultaneously transmitted by the 1394 cable 12 to a television, a computer (with a monitor) and a video-recorder, the television, the computer, and the video-recorder can each be programmed individually to show the first, second, and third movie respectively. Hence, by providing the parameter "chanID," the desired data are directed to be transmitted over a particular channel for showing on devices which have been programmed to receive that particular channel.

[64] The parameter "objectID" is used to identify the particular object or file which contains the desired data to be transmitted via the channel specified by the parameter "chanID."

[65] The parameter "speed" is used to specify the data transmission speed along the 1394 cable 12 for data transfer between the storage media and a display device. In the case of protocol 1394, the various alternative values that can be designated for the "speed" parameter include S100, S200 and S400.

[66] The parameter "type" as used in connection with the Play function call specifies how the play operation is to be performed on an incremental quantitative basis. In one embodiment, the alternative categories that can be specified as "type" are byte count or time. The use of the parameter "type" will become clearer when discussed in connection with the parameters "inPoint" and "outPoint" below.

[67] The parameter "inPoint" is used to specify the start position for the play operation. This start position can be specified in alternative ways depending on the parameter "type." If the parameter "type" is selected as count, then the start position can be specified using the number of sectors. It should be remembered that storage media such as hard disks are organized based on sectors. On the other hand, if the parameter "type" is selected as time, then the start position can be specified using time.

[68] The parameter "outPoint" is used to specify the end position for the play operation. As discussed with the parameter "inPoint," the end position for the parameter "outPoint" can similarly be specified in alternative ways depending on the parameter "type."

[69] The Record function call is used to record a track or object onto the storage media. A sample format for this function call is as follows:

Record (chanID, objectID, inPoint, type, *cb)

[70] As mentioned above, in an exemplary embodiment where the 1394 cable is used to transmit data, the A/V file system 20 is capable of receiving one or more data streams from various devices via multiple separate channels. The parameter "chanID" is therefore used to identify which one of the multiple channels is to be recorded.

[71] The parameter "objectID" is used to identify the object or file which is to be used to store the recorded data.

[72] The parameter "type" serves an identical function as discussed in connection with the Play function call.

[73] The parameter "inPoint" specifies the start position for the record operation. Similarly, the start position can be specified in alternative ways depending on the parameter "type" as discussed with the Play function call above.

[74] The Stop function call is used to stop a track or object which may be under recording or playing. A sample format for this function call is as follows:

Stop (chanID, *cb, *lba)

[75] Similarly, the parameter "chanID" is used to identify a specified channel so as to allow the A/V file system 20 to halt the record or play operation for that specified channel.

[76] The parameter "*lba" is a pointer used to return the current byte position from the A/V file system 20 to the A/V controller 18.

[77] The Pause function call is used to pause the play or record operation. A sample format for this function call is as follows:

Pause (chanID, *cb, *lba)

[78] The parameter "chanID" is used to identify a specified channel so as to allow the A/V file system 20 to pause the play or record operation for that specified channel.

[79] Similarly, the parameter "*lba" is a pointer used to return the current byte position from the A/V file system 20 to the A/V controller 18. The information

provided by this pointer can be used for various purposes such as speeding up the Resume function call.

[80] The Resume function call is used to restart the play or record operation. A sample format for this function call is as follows:

Resume (chanID, *cb)

---

[81] The parameter "chanID" is used to identify a specified channel so as to allow the A/V file system 20 to deactivate the pause operation for that specified channel.

[82] The GetCurrentLBA function call is used to retrieve the current offset or logical block address (in sector count) for the recorded track or object. A sample format for this function call is as follows:

GetCurrentLBA (chanID, *p_count)

---

[83] The parameter "*p_count" represents a pointer which points to a location that stores the count value assigned by the A/V file system 20 after an object or track has been recorded. This count value is used to update the HMS table. The A/V controller 18 supplies the parameter "*p_count" so that when the A/V file system 20 calls back after the function is performed, *p_count will contain the count value. Using the count value stored in *p_count by the A/V file system 20, the A/V controller 18 is then able to calculate the logical block address and access any data point within the recorded object.

[84] The second group of function calls further includes additional function calls which allow certain trick plays to be performed, including the FastForward function call, the SlowForward function call, the StepForward function call, the FastReverse function call, the SlowReverse function call, and the StepReverse function call.

[85] The FastForward function call is used to fast forward a data stream being carried on a specified channel. A sample format for this function call is as follows:

FastForward (chanID, type, interval, repeat, *cb)

---

[86] The parameter "chanID" is used to identify a specified channel so as to allow the A/V file system 20 to apply the fast forward operation to data being carried on that specified channel.

[87] The parameter "type" is used to specify how the fast forward operation is to be performed. In an exemplary embodiment, the data is encoded in MPEG format. Under the MPEG format, each frame of the data can be independently encoded using one of three different encoding algorithms, namely, the intracoding (I) algorithm, the predictive (P) algorithm, and the bidirectional (B) algorithm. Hence, a data stream represented by the encoding algorithm being used on a frame-by-frame basis may look like this, IBB PBB IBB PBB. The fast forward operation is then performed based on how the frames in the data stream are to be processed, which is specified by the parameter "type". For example, using the data stream given above, the parameter "type" having a value of "I" indicates that only frames which use the intracoding algorithm are to be processed by the A/V file system 20 and all other frames are to be ignored. Similarly, the parameter "type" having a value of "IP" indicates that the first processed frame must use the intracoding algorithm, the next processed frame must use the predictive algorithm, and the third and fourth processed frames must use the intracoding and predictive algorithms respectively, and so on for the entire data stream. Other possible values for the parameter "type" include "IPP," "IPPP," "no B" and so on. Based on the disclosure provided herein, a person of ordinary skill in the art will understand how to implement the different values for the parameter "type."

[88] The parameter "interval" is used to specify how much of a data stream is to be skipped before it is processed again. The value of this parameter is given in time; alternatively, the value can be given in the number of frames which can then be converted to time as necessary.

[89] The parameter "repeat" is used to specify the duration of display of a frame or the number of times that a frame is to be displayed in a data stream. In other words, for a frame to be displayed, this parameter specifies how many times an individual frame is to be displayed during the fast forward operation. The repeated display of an individual frame affects the visual output quality of the data stream during the fast forward operation.

[90] The SlowForward function call is used to forward a data stream being carried on a specified channel on a frame-by-frame or other specified incremental basis, with each frame being displayed at a specified frequency. A sample format for this function call is as follows:

SlowForward (chanID, repeat, increment, *cb)

---

[91] Similarly, the parameter "chanID" is used to identify a specified channel so as to allow the A/V file system 20 to apply the slow forward operation to data being carried on that specified channel and the parameter "repeat" is used to specify the duration of display or the number of times a frame is to be displayed in a data stream. The parameter "increment" is used to specify the incremental value which determines how frames in the data stream are to be processed during the slow forward operation. For example, by specifying the parameter "increment" as "1", the data stream is forwarded on a frame-by-frame basis; or by specifying the parameter "increment" as "2", the data stream is forwarded by skipping every other frame.

[92] The StepForward function call is used to forward a data stream being carried on a specified channel by one or more frames and then return the data stream to a pause state. A sample format for this function call is as follows:

StepForward (chanID, increment, *cb)

---

[93] Likewise, the parameter "chanID" is used to identify a specified channel so as to allow the A/V file system 20 to apply the step forward operation to data being carried on that specified channel. The parameter "increment" is used to specify the incremental value which represents the number of frames that are to be skipped before returning the data stream to the pause state.

[94] The three reverse operation function calls, FastReverse (), SlowReverse () and StepReverse (), are similar to the three forward operation function calls described above, except that the direction of data stream movement caused by the three reverse operation function calls is backward as opposed to forward. The sample formats for the three reverse operation function calls are as follows:

FastReverse (chanID, type, interval, repeat, *cb)
SlowReverse (chanID, repeat, increment, *cb)
StepReverse (chanID, increment, *cb)

---

[95] The various parameters for the three reverse operation function calls are similar to the parameters described in connection with the three forward operation function calls.

[96] Referring to Fig. 1, it should also be noted that the API of the present invention can be implemented to provide an interface between the SBP-2 controller 22 and the A/V file system 20. Following the description as provided herein, a person of ordinary skill in the art will know of ways, techniques, and methods to implement the present invention as an interface between the SBP-2 controller 22, as well as other devices, and the A/V file system 20.

[97] As mentioned above, packets transported using the SBP-2 protocol are delivered to the SBP-2 controller 22 for processing. If the SBP-2 controller 22 determines that the A/V file system 20 is to be invoked, the SBP-2 controller 22 similarly uses the appropriate API as described above to interact with the A/V file system 20.

[98] The function calls of the API under the present invention are capable of passing input and output parameters. It should be understood that these parameters can be passed in various different manners. For example, these parameters can be passed directly in a function call; alternatively, these parameters can also be passed via use of pointers. A person of ordinary skill in the art will know of ways and methods to implement the passing of parameters.

[99] In an exemplary embodiment, the API of the present invention is implemented via software using C and C++. However, it should be understood that a person of ordinary skill in the art will know of additional computer programming languages as well as other ways, techniques, and methods to implement the present invention.

[100] A number of advantages is realized by the API of the present invention. For example, the API supports the handling of large and small files by the A/V file system 20. A number of function calls of the API, such as the LoadDescriptor(), StoreDescriptor() and the DeleteDescriptor(), are designed to handle descriptors, while other function calls, such as Play(), Record(), Stop(), Pause() and Resume(), are designed to handle A/V files. As previously mentioned, a descriptor is a relatively small file typically used to store and provide information on the properties of a specific object or a list of objects. By having function calls that are designed to handle descriptors and A/V files, the A/V file system is then capable of handling both large and small files thereby allowing A/V data files and other types of data files to be processed in an efficient manner.

[101] The API also supports a variety of features typically offered by an A/V file system 20, such as scalability to large disks, directory structures and delete, add and record operations. For example, the API includes function calls which use the objectID parameter and the descID parameter thereby allowing the implementation of scalable disks

and directory structures. Also, the API includes function calls such as DeleteDescriptor(), StoreDescriptor() and Record() thereby allowing the implementation of delete, add and record operations.

[102] Moreover, the API allows trick play operations to be performed by the A/V file system 20. For instance, the API includes function calls such as FastForward(), SlowForward(), StepForward(), FastReverse(), SlowReverse() and StepReverse().

[103] Furthermore, the API allows the A/V file system 20 to provide additional capabilities. For instance, the API allows several A/V data streams or the same A/V data stream to be operated on in a concurrent manner in a number of operations including Play(), Record(), Stop(), Pause() and Resume(). This is made possible by the use of the "chanID" and/or "objectID" parameters in a function call. As discussed above, in an exemplary embodiment where the 1394 cable 12 is used to transmit data, the 1394 cable 12 is able to accommodate one or more data streams simultaneously using multiple separate channels. As a result, each recipient device is capable of displaying one of the data streams via a particular channel depending on how such recipient device is programmed.

[104] In one example, if three different movies are concurrently transmitted by the 1394 cable 12 to a television, a computer (with a monitor) and a video-recorder respectively, the API of the present invention can be used to control the three different movies independently.

[105] In an alternative example, the same movie can be shown concurrently on the television, the computer and the video-recorder. Furthermore, the television, the computer and the video-recorder can be controlled in such a way that the same movie can be playing on each of these devices at different times. In other words, the three devices can be independently playing the same movie at the same time. Hence, by providing the parameters "chanID" and "objectID," different or the same data can be directed to be transmitted independently over different channels in a concurrent manner.

[106] Moreover, the API supports record and play operations starting from within an A/V file. For instance, the function calls, Play() and Record(), both accept as an input parameter the parameter, inPoint, which specifies the start position of the desired operation within the A/V file. Another parameter, outPoint, is used by the Play() function call to specify the end position of the operation within the A/V file. By having function calls that are capable of initiating an operation starting from within an A/V file, the performance of random data access is improved.

[107] In addition, the API optimizes disk access. As mentioned above, different function calls of the API are designed for descriptors and A/V files respectively. By providing the capability to handle descriptors and A/V files separately, disk access is optimized and the retrieval and handling of data is improved.

[108] Finally, the API permits devices each capable of handling a different type of data to communicate with an A/V file system. More specifically, the API can be used by an A/V controller and a SBP-2 controller to interface with the A/V file system using the 61883 protocol and the SBP-2 protocol respectively.

[109] It is further understood that the examples and embodiments described herein are for illustrative purposes only and that various modifications or changes in light thereof will be suggested to persons skilled in the art and are to be included within the spirit and purview of this application and scope of the appended claims. All publications, patents, and patent applications cited herein are hereby incorporated by reference for all purposes in their entirety.